retrofit2面试题

1. retrofit的原理

> 通过java接口以及注解来描述网络请求，并用动态代理的方式，在调用接口方法前后（before / after）注入自己的方法，before通过接口方法和注解生成网络请求的request，after通过client调用相应的网络框架（默认okhttp）去发起网络请求，并将返回的response通过converterFactorty转换成相应的数据model，最后通过calladapter转换成其他数据方式（如rxjava Observable）

2. 什么是代理模式

> 为其他对象提供一种代理以控制对这个对象的访问，它主要解决在直接访问对象时带来的问题，比如说：要访问的对象在远程的机器上。在面向对象系统中，有些对象由于某些原因（比如对象创建开销很大，或者某些操作需要安全控制，或者需要进程外的访问），直接访问会给使用者或者系统结构带来很多麻烦，我们可以在访问此对象时加上一个对此对象的访问层

3. retrofit为什么要使用动态代理

> 因为1个静态代理 只服务1种类型的目标对象
> 若要服务多类型的目标对象，则需要为每种目标对象都实现一个静态代理对象 ，在目标对象较多的情况下，若采用静态代理，则会出现 静态代理对象量多、代码量大，从而导致代码复杂的问题，而动态代理只需要1个动态代理类就可以解决创建多个静态代理的问题，避免重复、多余代码
> 更强的灵活性设计动态代理类（DynamicProxy）时，不需要显式实现与目标对象类（RealSubject）相同的接口，而是将这种实现推迟到程序运行时由 JVM来实现
> 在使用时（调用目标对象方法时）才会动态创建动态代理类 & 实例，不需要事先实例化

4. 动态代理的缺点是什么?

> 效率低
> 相比静态代理中 直接调用目标对象方法，动态代理则需要先通过Java反射机制 从而 间接调用目标对象方法
> 应用场景局限
> 因为 Java 的单继承特性（每个代理类都继承了 Proxy 类），即只能针对接口 创建 代理类，不能针对类 创建代理类
> 即只能动态代理 实现了接口的类

5. retrofit是如何把接口转换为请求?

> 1.首先，通过method把它转换成ServiceMethod。
> 2.然后，通过serviceMethod，args获取到okHttpCall对象。
> 3.最后，再把okHttpCall进一步封装并返回Call对象。

JNI面试题

1. 谈谈你对 JNI 和 NDK 的理解

> JNI：
> JNI 是 Java Native Interface 的缩写，即 Java 的本地接口。
> 目的是使得 Java 与本地其他语言（如 C/C++）进行交互。
> JNI 是属于 Java 的，与 Android 无直接关系。
> NDK：
> NDK 是 Native Development Kit 的缩写，是 Android 的工具开发包。
> 作用是更方便和快速开发 C/C++ 的动态库，并自动将动态库与应用一起打包到 apk。

> NDK 是属于 Android 的，与 Java 无直接关系。
> 总结：
> JNI 是实现的目的，NDK 是 Android 中实现 JNI 的手段。

2. 谈谈你对 JNIEnv 和 JavaVM 理解

> JavaVM
> JavaVM 是虚拟机在 JNI 层的代表。
> 一个进程只有一个 JavaVM。（重要！）
> 所有的线程共用一个 JavaVM。（重要！）
> JNIEnv
> JNIEnv 表示 Java 调用 native 语言的环境，封装了几乎全部 JNI 方法的指针。
> JNIEnv 只在创建它的线程生效，不能跨线程传递，不同线程的 JNIEnv 彼此独立。（重要！）
> 注意：
> 在 native 环境下创建的线程，要想和 java 通信，即需要获取一个 JNIEnv 对象。我们通过 AttachCurrentThread 和 DetachCurrentThread 方法将 native 的线程与 JavaVM 关联和解除关联。

3. 解释一下 JNI 中全局引用和局部引用的区别和使用

> 全局引用
> 通过 NewGlobalRef 和 DeleteGlobalRef 方法创建和释放一个全局引用。
> 全局引用能在多个线程中被使用，且不会被 GC 回收，只能手动释放。
> 局部引用
> 通过 NewLocalRef 和 DeleteLocalRef 方法创建和释放一个局部引用。
> 局部引用只在创建它的 native 方法中有效，包括其调用的其它函数中有效。因此我们不能寄望于将一个局部引用直接保存在全局变量中下次使用（请使用全局引用实现该需求）。
> 我们可以不用删除局部引用，它们会在 native 方法返回时全部自动释放，但是建议对于不再使用的局部引用手动释放，避免内存过度使用。
> 扩展：弱全局引用
> 通过 NewWeakGlobalRef 和 DeleteWeakGlobalRef 创建和释放一个弱全局引用。
> 弱全局引用类似于全局引用，唯一的区别是它不会阻止被 GC 回收。

4. JNI 线程间数据怎么互相访问

> 考察点和上体类似，线程本来就是共享内存区域的，因此我们需要使用 全局引用。

5. 怎么定位 NDK 中的问题和错误

> 一般在开发阶段的话，我们可以通过 log 来定位和分析问题。
> 如果是上线状态（即关闭了基本的 log），我们可以借助 NDK 提供的 addr2line 工具和 objdump 工具来定位错误。详情：
> so 动态库崩溃问题定位（addr2line与objdump）
> 其它还可以使用 C/C++ 的一些分析工具。

6. 静态注册和动态注册

> 静态注册：
> 通过 JNIEXPORT 和 JNICALL 两个宏定义声明，Java + 包名 + 类名 + 方法名 形式的函数名。不好的地方就是方法名太长了。
> 动态注册：
> 通常在 JNI_OnLoad 方法中通过 RegisterNatives 方法注册，可以不再遵从固定的命名写法（当然为了代码容易理解，名称还是尽量和 Java 中保持一致）。

自定义view面试题

1. View绘画机制

> View的绘制主要涉及三个方法：onMeasure()、onLayout()、onDraw()
> onMeasure主要用于计算view的大小，onLayout主要用于确定view在ContentView中的位置，onDraw主要是绘制View。
> 在执行onMeasure()、onLayout()方法时都会通过相应的标志位或者对应的坐标点来判断是否需要执行对应的函数，如我们经常调用的invalidate方法就只会执行onDraw方法，因为此时的视图大小和位置均未发生变化，除非调用requestLayout方法完整强制进行view的绘制，从而执行上面三个方法。

2. 事件传递机制，如何处理滑动冲突

> 当手指触摸到屏幕时，系统就会调用相应View的onTouchEvent，并传入一系列的action。
> dispatchTouchEvent的执行顺序为：
> 首先触发ACTIVITY的dispatchTouchEvent,然后触发ACTIVITY的onUserInteraction
> 然后触发LAYOUT的dispatchTouchEvent，然后触发LAYOUT的onInterceptTouchEvent
> 这就解释了重写ViewGroup时必须调用super.dispatchTouchEvent();

3. Android如何弹性滑动对象

> 使用Scroller 调用scrollTo/scrollBy配合View#computeScroll来完成

4. Activity,Window,View三者的联系和区别?

> Activity像一个工匠（控制单元）
> Window像窗户（承载模型）
> View像窗花（显示视图）
> LayoutInflater像剪刀
> Xml配置像窗花图纸。

5. 自定义View执行invalidate()方法,为什么有时候不会回调onDraw()

> 自定义一个view时，重写onDraw。调用view.invalidate(),会触发onDraw和computeScroll()。前提是该view被附加在当前窗口.
> view.postInvalidate(); //是在非UI线程上调用的
> 自定义一个ViewGroup，重写onDraw。onDraw可能不会被调用，原因是需要先设置一个背景(颜色或图)。表示这个group有东西需要绘制了，才会触发draw，之后是onDraw。因此，一般直接重写dispatchDraw来绘制viewGroup.自定义一个ViewGroup,dispatchDraw会调用drawChild.